라이더 기하학을 이용한 효율적인 차선 탐지 방법

최은서○ 김강희+
숭실대학교 AI융합학부
eunseo.choi.d@gmail.com, khkim@ssu.ac.kr

A Lane Detection Method Using a Lidar Geometry

Eunseo Choi○ Kanghee Kim+
Soongsil University

요 약

차선 탐지는 자율주행의 핵심적인 기능으로서 자차에 대한 고정밀 측위가 불가능한 경우에도 차로 유지 기능을 가능하게 한다. 일반적으로 차선 탐지는 카메라 이미지를 이용한 영상 처리에서 광범위하게 연구되었으나, 3차원 라이더 센서로부터 얻은 라이더 스캔을 이용하는 방법들이 연구되고 있다. 라이더 스캔은 라이더 빔들이 사물들에 충돌한 점들의 집합, 즉 포인트 클라우드 형식인데, 각 포인트는 3차원 좌표 외에도 반사광의 세기(intensity) 정보를 갖고 있기 때문에, 이 정보로부터 차선을 식별할 수 있다. 본 논문은 라이더 기하학을 이용한 효율적인 차선 탐지 방법을 제안한다. 3차원 라이더의 기하학을 이용하면, 많은 GPU 자원을 요구하는 카메라 기반 차선 탐지 방법들과 다르게, 적은 CPU 자원을 이용해서 효율적으로 차선을 탐지할 수 있다. 제안하는 방법은 자율주행 실험도시 K-CITY에서 수집한 라이더 스캔들을 이용하여 평가하였으며, 평가 결과에 따르면 평균 21 ms의 적은 CPU 시간만으로 효과적인 차선 탐지가 가능함을 확인할 수 있었다.

1. 서 론

차선 탐지는 차로 유지를 위해서 자율주행 레벨 2에서부터 요구되는 핵심적인 기능이다. 차선 탐지는 고정밀 측위가 불가능한 경우들, 예를 들어 GNSS 센서의 경우에는 터널 구간과 고층빌딩 구간들, 라이더(Lidar) 센서의 경우에는 고정밀 지도와 대조할 랜드마크들이 부족한 구간들을 통과할 때, 자율주행차가 필수적으로 의지해야 하는 기능이다.

라이더 센서는 일정한 주기마다 라이더 스캔을 수집하며, 라이더 빔들이 사물들에 충돌한 점들의 집합, 즉 포인트 클라우드(point cloud)를 반환하는데, 각 포인트는 3차원 좌표 외에도 반사광의 세기(intensity) 정보를 가지고 있다. 이 반사광 세기는 도로면의 차선들에 대해서는 값이 상대적으로 크기 때문에, 반사광 세기를 관찰하면 차선을 식별할 수 있다. 또한, 라이더는 여러 개의 광원을 360도 회전시키며 빔을 쏘기 때문에, 노면에서 원 형태의 궤적을 만들어내는데, 이 궤적의 z 값을 관찰하면 대상 노면이 주행 가능한 구간인지 아닌지를 식별할 수도 있다. 요약하면, 라이더 기하학을 이용하면 노면 분리와 차선 후보점 탐색을 용이하게 수행할 수 있다.

차선 탐지는 카메라 이미지를 이용한 영상 처리 분야에서 광범위하게 연구되어왔다.[1] 그러나 카메라 기반 차선 탐지는 라이더 기반 탐지와 비교할 때, 훨씬 많은 계산량을 요구한다. 예를 들어, 자율주행차에서 많이 사용되는 FHD 해상도(1920×1080)의 카메라는 매 이미지마다 약 2백만 화소를 반환하는데, 이는 64채널×1,800빔 해상도의 라이더가 매 스캔마다 약 11만 포인트들을 반환하는 것과 비교할 때, 약 20배 수준으로 많은 데이터이다. 따라서 카메라 기반 차선 탐지는 GPU와 같이 값비싼 고성능 병렬 처리기를 요구하는 단점이 있다.

본 논문은 라이더 기하학[2]을 이용한 차선 탐지 방법을 제안한다. 라이더 기하학을 이용하면, 주행 가능한 노면을 분리하고, 그 안에서 차선 후보점을 탐색하는 것을 용이하게 수행할 수 있다. 또한 라이더 스캔은 카메라 이미지에 비해서 데이터량이 훨씬 적기 때문에, CPU 자원만으로도 효과적으로 차선 탐지를 수행할 수가 있다. 반면에 라이더 스캔의 적은 데이터량은 원거리 차선을 탐지하는 데는 불리하게 작용할 수 있다. 차선은 보통 15cm 폭을 갖는데, 하나의 라이더 채널은 정해진 개수의 빔을 일정한 회전각을 가지고 방사형으로 발사하기 때문에 빔이 충돌한 노면이 멀리 있을수록 그 노면의 차선을 샘플링할 가능성이 낮아진다. 본 논문은 라이더 기하학을 이용하여, 라이더의 각 채널에 대해서 차선 후보점들을 추출하고, 이 후보점들에 대해서 RANSAC 알고리즘과 가중 이동 평균을 적용하여 차선 곡선을 유도함으로써, 높은 정확도로 차선을 추출할 수 있음을 보인다. 제안하는 방법은 전체 구간이 약 1.6 km에 달하는 자율주행 실험도시 K-CITY에서 수집한 라이더 스캔들을 이용해서 평가한 결과, 차선이 일부 끊어지는 사거리 구간과 차로를 변경하는 경우에도 연속적인 차선 추출이 가능함을 확인하였다. 또한, 매 100 ms마다 수집된 라이더 스캔에 대해서 최대 31 ms의 CPU 시간만 요구함을 확인하였다. 이는 라이더가 카메라보다 데이터량이 훨씬 적음에도 불구하고, 차선을 탐지하는데 효과적인 센서임을 의미하는 것이라 볼 수 있다.

2. 관련 연구

차선 탐지는 카메라를 이용한 방법들과 라이더를 이용한 방법들로 나뉜다. [1]에서는 카메라를 이용하는 차선

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2021R1A2C2012837, NRF-2021R1A4A1032252).

에지 제안 단계와 차선 추출 단계로 구성되는 LaneNet을 제안하여, 96% 이상의 TPR (True Positive Rate)을 달성하였다. [2]에서는 라이더 기하학을 이용하여 차량 지붕에 부착된 센터 라이더를 이용하여, 효과적인 노면 분리와 차선 탐지가 실시간으로 가능함을 보였다. [3]에서는 라이더 기반 탐지 결과와 카메라 기반 탐지 결과를 칼만 필터로 융합하여 약 25 Hz의 탐지 성능을 얻었다. [4]에서는 고정밀 지도를 제작할 목적으로 라이더 기반 차선 탐지를 수행하였는데, 차로들이 평행하다는 조건을 이용하여 차선 탐지의 정확성을 높였다. 본 논문은 라이더 기하학[2]을 이용한 차선 탐지를 수행하며, 차선 탐지의 정확성을 높이기 위해서 최근 n개의 라이더 스캔을 병합[3]하고, 차로 평행 조건[4]을 함께 적용한다.

3. 라이더 기하학을 이용한 차선 탐지

제안하는 방법은 높은 정확도로 차선을 탐지하기 위해 포인트의 위치 정보를 활용하여 주행 가능 구간을 검출한 다음, 구간 안에서 반사광 세기를 활용하여 차선 후보점들을 검출하고 RANSAC과 가중이동평균을 적용한다.

3.1. 포인트 클라우드들의 병합

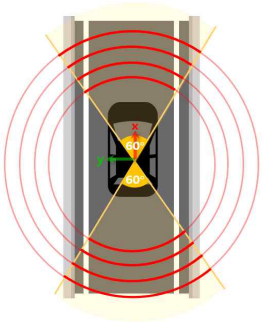


그림 1. 관심 영역에 속하는 포인트들 검출

과거에 수집한 포인트 클라우드를 누적해서 포인트를 밀집시킴으로써 차선에 포인트가 명중할 확률을 높인다.

100ms를 주기로 포인트 클라우드가 수신되면, 그림 1과 같이 먼저 차량의 진행 방향 기준으로 전방과 후방 60 ° 관심 영역에 속하는 포인트들을 검출한다. 다음으로, 차량으로부터 얻은 선속도와 각속도 정보를 이용하여 과거부터 현재까지 이동한 변위를 계산한다. 이를 바탕으로 과거에 수집한 포인트 클라우드들의 좌표계를 모두 현재를 기준으로 변환한 뒤, 현재 시점에서 얻은 포인트 클라우드와 병합한다.

3.2. 주행 가능 구간 검출

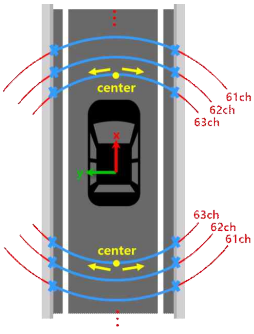
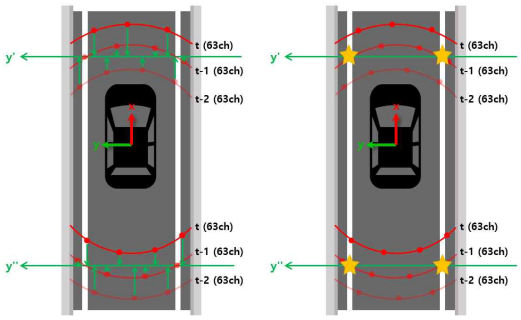


그림 2. 주행 가능한 구간 검출

그림 2와 같이 연석과 가드레일 등 도로에서 높이 편차가 존재하는 지점을 경계로 하여, 포인트의 z 값을 기준으로 주행 가능한 구간을 검출한다. 이때 사용하는 좌표계는 현재 차량의 위치를 원점으로 하고, 차량의 헤딩을 x축, 그에 수직인 축을 y축으로 한다.

주행 가능한 구간을 검출하기 위해 채널마다 범위를 나누고, 각각의 범위에 대해 표준편차를 계산한다. 자차와 가장 근접한 63번 채널에서 y 값이 0인 지점부터 전방과 후방에 대해서 좌우로 탐색을 진행하다가, z 값의 표준편차가 일정 값 이상인 지점을 만나면 탐색을 멈춘다. 검출된 양 끝 지점 사이에 속하는 포인트들을 검출한다. 다음으로 63번 채널을 제외한 다른 채널에서는 직전 채널에서 검출된 양 끝 포인트의 중앙을 시작 지점으로 설정하고, 앞선 방법을 반복하여 진행한다.

3.3. 차선 후보점 검출



(a) (b)

그림 3. y축과 평행한 축에 포인트들 투영하기(좌측)와 단일 채널에서 검출된 4개의 차선 후보점들(우측)

서로 다른 포즈에서 수집된 포인트 클라우드들을 누적시키면 같은 채널 번호인 링들이 여러 개 생긴다. 이때 같은 채널에 속하는 모든 포인트들을 한 번에 탐색하기 위해서 그림 3(a)와 같이 y축과 평행한 축에 포인트들을 투영시킨다. 채널마다 투영시킨 포인트들에 대해 범위를 나누고, 각각의 범위에 대해서 반사광의 세기의 표준편차를 계산하여 차선 후보점을 구한다. 위 과정을 차량의 전방과 후방에 대해서 각각 진행하게 되면, 차선 후보점은 누적된 채널마다 그림 3(b)와 같이 차량의 전방과 후방의 좌 우에서 총 4개 검출된다.

3.4. 차선 모델 검출

차선 모델은 왼쪽과 오른쪽 차선의 방정식 묶음으로 구성되어 있다. 먼저 왼쪽 차선을 기준으로 한 모델을 구하기 위해, 왼쪽 차선 후보점들 중에서 3개를 무작위로 샘플링하고, 샘플을 이용하여 이차 곡선 방정식을 $y = ax^2 + bx + c_1$ 형태로 구한다.[3] 이때, 두 개의 차선이 서로 평행하다고 가정하면, 왼쪽 차선의 방정식을 차로의 폭만큼 평행이동하여[4] 오른쪽 차선의 방정식을 $y = ax^2 + bx + c_2$ 형태로 구할 수 있다. 이러한 과정을 오른쪽 차선 후보점에 대해서도 동일하게 수행하여 오른쪽 차선을 기준으로 한 모델을 구한다. RANSAC 알고리즘을 적용하여, 전체 차선 후보점들에 대해 가장 많은 후보점들의 지지를 받는 모델을 검출한다.

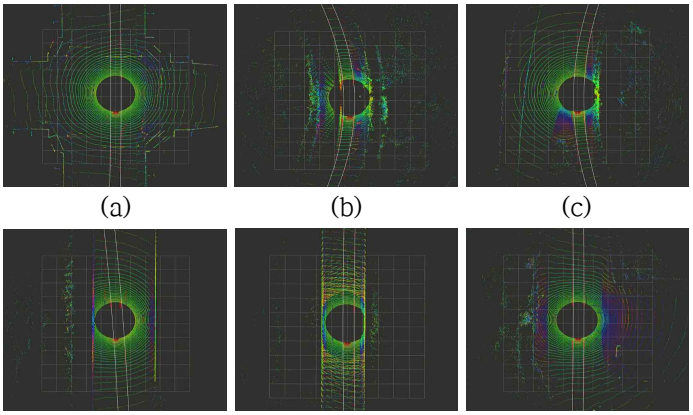
3.5. 가중 이동 평균

RANSAC 알고리즘만 사용할 경우 차선 후보점의 수가 부족하거나, 이상치 데이터가 포함되어 모델의 정확도가 떨어질 수 있다. 따라서 신뢰도가 높은 모델에 더 큰 가중치를 부여하는 가중이동평균을 적용해 높은 정확도를 보장한다.

4. 실험 결과

제안하는 차선 탐지 알고리즘은 Ubuntu 20.04, ROS2 Galactic 버전, PCL(Point Cloud Library) 1.10 버전을 이용하여 구현하였다. 그리고 성능 평가를 위해 경기도 화성시에 위치한 자율주행 실험도시 K-CITY에서 1.6 km의 루프 구간에 대해서 64 채널의 OS1-64 라이더를 이용하여 수집한 라이더 스캔들을 이용하였다. 실험에 사용한 PC는 12코어 인텔 CPU와 16GB 메모리를 가진다.

그림 4. K-CITY 실험 구간



(a) (b) (c)

(d) (e) (f)

그림 5. 교차로 구간(a), 곡선 구간(b), 램프 구간(c), 차로 변경 구간(d), 터널 구간(e), 직선 구간(f)

그림 4는 실험을 진행한 K-CITY의 구간별 특징을 설명한 그림이다. 그림 5는 구간별 차선 탐지 결과에 대한 그림이다. 그림 5(a)는 차선 후보점의 수가 충분치 않고 노이즈로 작용하는 차로 유도선 등이 많은 교차로 구간, 그림 5(b)는 도로 바깥쪽에 풀숲이 존재하는 곡선 구간, 그림 5(c)는 다차로 도로로 합류하는 램프 구간, 그림 5(d)는 다차로 안에서 차로 변경을 시행하는 구간, 그림 5(e)는 터널 구간, 마지막으로 그림 5(f)는 직선 구간이다. 결과적으로 K-CITY 대부분의 구간에서 차선들이 잘 검출되었음을 확인할 수 있다.

탐지 거리 / 실험 구간	5m	10m	15m	20m	25m	30m
전체 구간	96.7%	96.5%	94.2%	88.6%	81.1%	71.2%
직선 구간	100.0%	100.0%	100.0%	99.6%	98.3%	95.8%
S자 구간	100.0%	100.0%	98.4%	82.4%	70.2%	51.6%

표 1. K-CITY 구간별 검출된 차선 모델 정확도

표 1은 구간별 차선 탐지 거리를 계산한 것이다. 차선 탐지 거리 d는 전방 d 미터 지점에서 계산된 차선 상의 포인트와 ground truth 포인트 간의 오차가 1 미터 미만임을 의미한다. 전체 구간에서는 탐지 거리 15m 이내에서, 모델 검출에 대한 정확도가 90% 이상으로 준수한 정확도를 보였다. 하지만 탐지거리가 20m 이상일 때 정확도가 떨어지는데, S자 구간에서 2개 이상의 변곡점이 생기는 경우에 2차 방정식이 차선 피팅에 한계를 보이기 때문이다. 직선 구간에서는 탐지거리 30m 이내의 전 영역에서 95% 이상의 높은 정확도를 보였다.

CPU 시간 / 차선 탐지 단계	최소(ms)	평균(ms)	최대(ms)
1. 포인트 클라우드 누적	1.88	5.03	12.33
2. 주행 가능 영역 검출	2.52	6.42	10.26
3. 차선 후보점 탐색	1.95	5.31	12.92
4. 차선 모델 검출	1.82	5.06	15.21
5. 가중 이동 평균	0.01	0.01	0.01
총 소요시간	13.15	21.76	31.44

표 2. 차선 탐지 단계별 소요 시간

표 2는 K-CITY 전체 구간에 대해 차선을 탐지하는 단계별로 소요 시간을 측정한 결과이다. CPU 시간은 평균적으로 약 21ms 소요되는 것을 확인했으며, 최대 약 31ms로 라이더의 포인트 클라우드 발행 주기인 100ms에 비해 훨씬 적은 소요 시간으로 차선 탐지를 할 수 있음을 확인하였다.

5. 결 론

본 논문은 라이더 기하학을 이용한 효율적인 차선 탐지 방법을 제안하였다. 3차원 라이더의 기하학을 이용하면, 적은 CPU 자원을 이용해서 효율적으로 차선을 탐지할 수 있다. 실험 결과에 따르면, 제안하는 방법은 평균 21 ms의 적은 CPU 시간만으로 효과적인 차선 탐지가 가능하였다. 향후 연구로, 자차가 주행하는 현재 차로뿐만 아니라 이웃 차로들을 추가로 탐지하고, 차선 후보점을 검출하는 작업을 병렬화할 계획이다.

참고문헌

[1] Z. Wang, W. Ren, and Q. Qiu, “LaneNet: Real-Time Lane Detection Networks for Autonomous Driving,” https://doi.org/10.48550/arXiv.1807.01726, 2018.

[2] F. Ghallabi et al, “LIDAR-Based Lane Marking Detection for Vehicle Positioning in an HD Map,” https://hal.archives-ouvertes.fr/hal-01891764, 2018.

[3] K. Burnett et al, “Building a Winning Self-Driving Car in Six Months,” International Conference on Robotics and Automation (ICRA), 2019.

[4] J. Jung and S.-H. Bae, “Real-Time Road Lane Detection in Urban Areas Using Lidar Data,” Electronics Vol. 7, No. 11, 2018.